Amendments to the specification,

Marked version of the replacement paragraph(s)/section(s), pursuant to 37 CFR 1.121(b)(1)(ii):

Please replace the section heading of the claims (i.e., "Claims" which was automatically generated by the PTO Electronic Stylesheet v. 1.1.1, pursuant to electronic filing), with the following rewritten heading:

~~Claims~~What is claimed is:

Please replace the title of the Application at page 1 with the following rewritten title:

~~Database System Providing High Performance Database Versioning~~System and Methodology for Performing Read-Only Transactions in a Shared Cache

Please replace paragraph [0010] with the following rewritten paragraph:

In order to provide support for read-only transactions, one of the things that is needed is that the database system should not block for things like database locks, such as a table lock or row lock. However, in current database systems locks are almost always used for write transactions to achieve the goal of serializing the transaction, thereby serving as an obstacle to supporting read-only transactions in these systems. There are some current systems that have limited support for read-only transactions of this nature. Unfortunately, these current systems have a number of drawbacks and limitations. For example, at least one database system, the InterBase® database system, uses a multi-generational architecture that avoids use of locks. However, the multi-generational approach has the disadvantage of the extra overhead required to maintain multiple database versions.

Please replace paragraph [0014] with the following rewritten paragraph:

Another current approach is that illustrated by a current Oracle® solution (available from Oracle of Redwood Shores, CA). The Oracle® solution stores back versions in the database file itself. A disadvantage of this approach is that certain read-only transactions will sometimes fail because not all back versions are guaranteed to be maintained forever. Basically, the problem is that in some cases these back versions will

overflow. When there are too many back versions, a certain number of back versions will be dropped. If a read-only transaction needs a particular back version and it has been dropped, then the read-only transaction will fail. Typically, the transaction will actually abort with an error message in the event of this type of failure. Given these shortcomings in current product offerings, a better solution is sought.

Please replace paragraph [0026] with the following rewritten paragraph:

ISAM: ISAM refers to the "Indexed Sequential Access Method" which is a disk storage and access method. In practice, the term ISAM is sometimes used to refer to desktop and/or file-based databases or IBM's Information Management System (IMS™) and Pervasive Btrieve® databases. It also is used to refer to navigational database applications that rely on a procedural approach to data access and retrieval. Under ISAM, records are generally located using a key value. A smaller index file stores the keys along with pointers to the records in the larger data file. The index file is first searched for the key and then the associated pointer is used to locate the desired record.

Please replace paragraph [0027] with the following rewritten paragraph:

JDBC: JDBC is an application-programming interface (API) that provides database access from the Java™ programming language. JDBC allows Java™ applications to access multiple database management systems. A set of interfaces is included in the standard JDBC API for opening connections to databases, executing SQL commands, and processing results. Each relational database management system usually requires a driver to implement these interfaces. A JDBC driver manager typically handles multiple drivers that connect to different databases. Accordingly, JDBC calls are generally sent to the JDBC driver manager, which passes the call to the driver for interacting with the specified database. For further information on JDBC, see e.g., "JDBC 3.0 API Documentation", from Sun Microsystems, the disclosure of which is hereby incorporated by reference. A copy of this documentation is available via the Internet (e.g., currently at java.sun.com/products/jdbc/download.html#corespec30).

Please replace paragraph [0033] with the following rewritten paragraph:

Referring to the figures, exemplary embodiments of the invention will now be described. The following description will focus on the presently preferred embodiment of the present invention, which is implemented in desktop and/or server software (e.g., driver, application, or the like) operating in an Internet-connected environment running under an operating system, such as the Microsoft Windows® operating system. The present invention, however, is not limited to any one particular application or any particular environment. Instead, those skilled in the art will find that the system and methods of the present invention may be advantageously embodied on a variety of different platforms, including Macintosh®, Linux®, Solaris™, UNIX®, FreeBSD®, and the like. Therefore, the description of the exemplary embodiments that follows is for purposes of illustration and not limitation. The exemplary embodiments are primarily described with reference to block diagrams or flowcharts. As to the flowcharts, each block within the flowcharts represents both a method step and an apparatus element for performing the method step. Depending upon the implementation, the corresponding apparatus element may be configured in hardware, software, firmware or combinations thereof.

Please replace paragraph [0054] with the following rewritten paragraph:

In basic system operation, clients typically connect to the database system (i.e., JDataStore) using JDBC drivers. The system includes support for two different types of JDBC drivers. As shown, a remote client 310 may connect via a remote JDBC driver (e.g., driver 320) which communicates over a network using TCP/IP. The system also includes an in-process JDBC driver 330 which calls directly on the same call stack in order to provide increased efficiency. The same API is used for both drivers; however the in-process JDBC driver calls on the same call stack while the remote JDBC driver marshals over TCP/IP. For further description of the basic operation/design of JDataStore, see "JDataStore 7 Developer's Guide" (Part No. JBE0090WW21004), available from Borland Software Corporation of Scott's Valley, CA, the disclosure of which is hereby incorporated by reference. A copy of this document is also available via the Internet (e.g., currently at (ftpc) ~~ftpc.~~ borland.com/pub/jdatastore/techpubs/jdatastore7/jds_devgd.zip).

Please replace paragraph [0099] with the following rewritten paragraph:

Then the last item to delete is the shadow table itself. Recall that the shadow table is very compact because there are only two columns in the shadow table. As a result the shadow table typically requires no more than 8 bytes for each block stored, and they are all sequential. Also, dropping tables in JDataStore for similar reasons is very fast. Thus, even if the shadow ~~cache~~ table did get larger it would still be deleted very, very quickly. The present invention does not require any complex garbage collection process, so this process can proceed much faster than prior solutions with complex garbage collection routines.

Please replace the Abstract paragraph at page58 with the following rewritten paragraph:

A database system providing high performance database versioning is described. In ~~one embodiment, for example, in~~ a database system employing a transaction log, ~~an improved~~ a method ~~of the present invention is described~~ for restoring databases to a consistent version~~, the method~~ comprises steps of: providing a shared cache storing database blocks for use by multiple databases; for a read-only transaction of a given database, creating a cache view of the shared cache using the given database's transaction log, the cache view comprising particular database blocks of the shared cache that record a view of a particular version of the database at a given point in time; creating a shadow cache for storing any database blocks that overflow the cache view; and in conjunction with the cache view and the shadow cache, preserving a logical undo operation for the read-only transaction of the given database, so as to allow the given database to be restored to a transactionally consistent version ~~upon starting the read-only transaction~~.